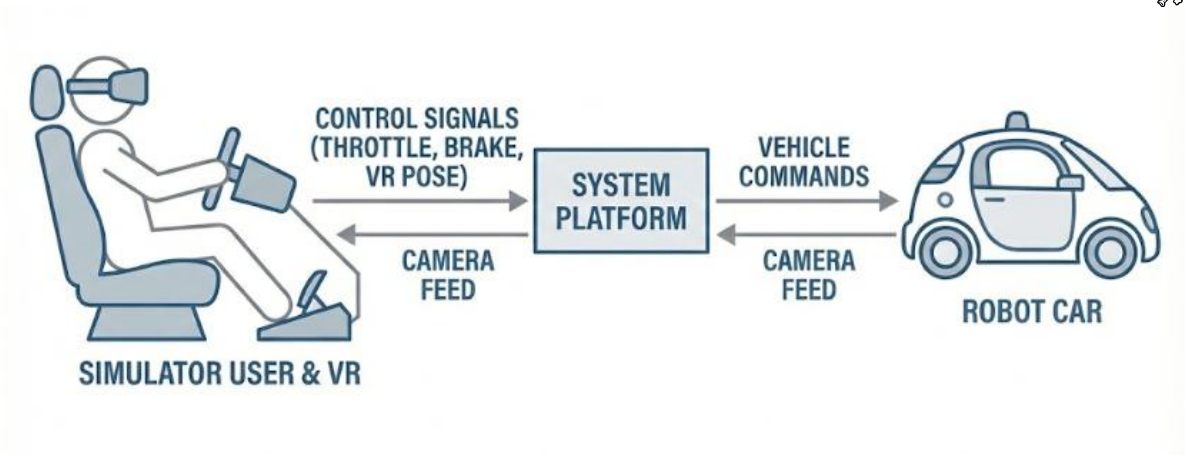
CHALMERS



Development of a Driver-in-the-Loop Advanced Driver Assistance Systems Prototyping Platform

TME180 Automotive Engineering Project 2025

Abhay Chouhan
Linus Fäldt
Elliot Jonsson
Fanxiang Liao
Prakash Raju Sridharraju
Jingyu Wang

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2025

Development of a Driver-in-the-Loop Advanced Driver Assistance System Prototyping
TME180 Automotive Engineering Project 2025

Supervisor: Marco Dozza, Department of Mechanical Engineering
Supervisor: Rahul Rajendra Pai, Department of Mechanical Engineering
Examiner: Alexey Vdovin, Department of Mechanical Engineering

Studentarbeten – Mekanik och maritima vetenskaper (M2) – Projektarbete
Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31 772 1000

Abstract

This project presents the development of a small-scale Driver-in-the-Loop Advanced Driver Assistance System prototyping platform based on a MentorPi robot vehicle and a modular Robot Operating System 2 software architecture. The platform integrates human driver inputs (steering wheel, pedals, and gear selection), onboard sensing (monocular camera and 360° LiDAR), and real-time visualisation into a closed-loop test setup for safe and repeatable indoor testing. Functionality is distributed across Robot Operating System-2 nodes for driver input handling, motion control, LiDAR processing, state aggregation, and live camera streaming with information overlays.

A simplified Automatic Emergency Braking function is implemented using an Enhanced Time-to-Collision, using a stopping-distance trigger. Rather than relying on a fixed distance threshold, this approach estimates whether the vehicle can safely stop before reaching an obstacle by accounting for the speed and braking capability, making the intervention more representative of practical safety behaviour. Two operating modes were validated: a baseline browser-based camera view and a virtual reality mode using a Meta Quest 3. In the VR configuration, the camera feed is accessed via an HTTP snapshot endpoint, while head-tracking data are transmitted over User Datagram Protocol to control the pan–tilt camera.

Results show stable baseline teleoperation, with 20 consecutive laps completed without system restart, and successful execution of a combined VR driving–slalom, Automatic Emergency Braking scenario in four out of five runs. However, VR operation occasionally led to a paralyzed control state, indicating integration and stability limitations under increased system load. Beyond functional validation, the platform is intended to enable rapid prototyping and early-stage evaluation of Active Safety and Advanced Driver Assistance Systems concepts. Its low cost, modular design, and safe indoor operation make it particularly suitable for pedagogical activities at Chalmers University of Technology, supporting hands-on learning and experimentation in courses related to Active Safety and Driver-in-the-Loop system development.

List of Acronyms

Below is the list of acronyms that have been used throughout this project, listed in alphabetical order:

ADAS	Advanced Driver Assistance Systems
AEB	Automatic Emergency Braking
DiL	Driver-in-the-loop
ETTC	Enhanced Time-to-Collision
hDiL	Human Driver in the Loop
HMI	Human-Machine Interaction
LiDAR	Laser imaging, detection and ranging
ROS	Robot Operating System
UDP	User Datagram Protocol
VR	Virtual Reality

Nomenclature

Below is the parameters and variables that have been used throughout this project.

Parameters

$v_{\max}$	Maximum allowed vehicle speed used to scale driver input
L	Effective wheelbase of the robotic vehicle used in steering kinematics
$a_{\max}$	Assumed maximum achievable deceleration during emergency braking
t_{reaction}	Assumed system reaction time used in stopping distance calculation
C	Servo center position for camera pan–tilt control
R	Servo actuation range used to map head rotation to PWM signals
$\alpha_{\max}$	Maximum allowed camera rotation angle

Variables

u_{steer}	Normalized steering input, $u_{\text{steer}} \in [-1, 1]$
u_{throttle}	Normalized throttle input, $u_{\text{throttle}} \in [0, 1]$
u_{brake}	Normalized brake input, $u_{\text{brake}} \in [0, 1]$
u_{drive}	Combined longitudinal drive command after brake suppression
g	Discrete gear state (forward, neutral, reverse)
v	Longitudinal vehicle velocity command
ω	Vehicle yaw rate command
R	Instantaneous turning radius of the vehicle
δ	Steering angle used in kinematic approximation
$d_{\min}$	Minimum measured obstacle distance in the forward LiDAR sector
d_{stop}	Computed stopping distance used by AEB logic
α	Measured head rotation angle from VR headset

Contents

1
Introduction

Road traffic crashes remain a major public-health problem, with increasing fatalities worldwide. Many crashes are linked to human errors such as distraction, misjudgment, or delayed reactions, which means that the driver is both part of the problem and part of the solution. Active safety systems aim to support the driver before a crash occurs by detecting critical situations early and helping the driver avoid or mitigate them [1].

Advanced Driver Assistance Systems (ADAS) are a family of active safety functions operating at low to medium levels of automation, where the human retains primary driving responsibility [2, 3]. Examples include warning systems such as Forward Collision Warning and intervention systems such as Automatic Emergency Braking (AEB) or Adaptive Cruise Control (ACC), which either inform the driver, prepare the vehicle, or briefly intervene [3]. In all these cases, automation does not replace the driver; instead, it extends the driver's operational safety margin when human performance is temporarily insufficient.

To study this shared-control relationship, a Driver-in-the-Loop (DiL) platform provides a realistic yet controlled experimental environment [4]. In a DiL setup, the driver remains fully responsible for tactical and strategic decisions, while ADAS functions continuously monitor the environment and support the driver through warnings, alerts, or limited interventions. This enables systematic investigation of how drivers perceive and react to safety-critical feedback, how effectively ADAS functions communicate risk, and how human decision-making adapts when automation is active but not dominant.

By keeping the human actively involved rather than removing them from the control loop, DiL platforms form a crucial bridge between purely automated testing and real-world driving, allowing safety functions to be evaluated not only in terms of technical performance but also in terms of human interaction, trust, and behavioral response [4].

In this project, a small-scale robot vehicle was transformed into a DiL ADAS prototyping platform using the Robot Operating System 2 (ROS 2) as the integration framework. The platform combines steering wheel and pedal input, camera and LiDAR sensing, and real-time visual feedback—including a VR mode—allowing a human operator to drive the robot while an AEB function monitors longitudinal threats using a simplified Enhanced Time-to-Collision concept [5]. The objective is

not full automation, but a reproducible setup in which human involvement, low-level automation limitations, and basic active safety behavior can be explored safely in an indoor environment.

1.1 Problem Statement

This project aims to design and implement a modular DiL ADAS prototyping platform on a small-scale robotic vehicle, with a focus on system integration, HMI, and functional validation. Many existing platforms rely on coupled, hardware-specific implementations, making experiments difficult to reproduce or extend. To address this, an integrated ROS 2-based DiL platform is developed that:

- Streams a real-time, annotated camera view to remote operators while publishing LiDAR-based distance information and hazard warnings with low latency.
- Provides safe and configurable driver control interfaces, including joystick, keyboard, and steering wheel with pedals.
- Maintains a modular and extensible architecture, allowing sensors and control interfaces to be replaced or reused in future experiments.

1.2 Project Scope

The project was subject to several practical and methodological constraints summarised in Table 1.1, but the chosen methodology still provides a robust, scalable base for future extensions of the DiL ADAS prototyping platform. The platform is designed for rapid prototyping of active safety functions in a controlled environment, making it well-suited for early development and evaluation of ADAS in pedagogic activities at Chalmers, for example, in courses like TME192 Active Safety, where students can experiment with DiL scenarios on real hardware.

Table 1.1: Project scope and limitations

Category	Description
Time limitation	The total duration of 20 weeks allows for a functional prototype but not a full-scale research platform.
Personnel limitation	The team consists of six students, supported by one professor and one supervisor. Everyone contributes part-time alongside other courses, requiring efficient coordination and scheduling.
Financial limitation	Work was performed exclusively with equipment supplied by the course. To remain cost-efficient, all experiments used the standard MentorPi robot, without extra external equipment.
Testing limitation	All tests will be done indoors under controlled conditions.
Algorithm limitation	ADAS functions are implemented as simplified logics, without machine-learning or perception algorithms.
Sensor limitation	The robot's single-camera and single-LiDAR configuration restricts environmental coverage and sensor redundancy.

2
Background

From an engineering perspective, active safety systems can be understood as a closed-loop process that transforms sensor information into actionable driver information or vehicle actions in real-time. This process (figure: 2.1) typically spans three stages: environment sensing, assessment of risk using decision logic and algorithms, and interaction with the driver and vehicle through the HMI and actuators. System performance depends not only on the individual components, but also on how reliably and consistently this chain operates as an integrated whole [1, 3, 6].

Figure 2.1: Active safety process

Evaluating this process requires platforms that enable the simultaneous study of sensing, decision-making, and driver interaction under realistic yet controlled conditions. DiL setups are therefore a practical engineering tool, as they enable functional validation of active safety concepts while explicitly accounting for human behaviour, interface design, and system timing effects. Such platforms are particularly relevant during early development, where modularity, reproducibility, and ease of experimentation are more important than system optimality.

This section introduces the technical background required to understand the proposed DiL ADAS platform. It outlines the robotic vehicle hardware, the ROS 2–based software architecture enabling modular integration, and the role of the driver and HMI within the active safety process.
Advanced Driver Assistance Systems (ADAS) encompass a wide range of vehicle technologies designed to enhance road safety by assisting the driver through perception, decision support, and limited automated intervention. These systems rely on complementary sensors and computational modules to detect hazardous situations and reduce crash risk [2, 3, 6].

2.1 Driver Interface Devices

The DiL platform relies on external HMI devices to enable realistic driver interaction. In this project, driver input and perception are primarily mediated through a force-feedback steering wheel with pedals and a head-mounted VR display.

A Logitech steering wheel and pedal set is used to capture continuous steering, throttle, and braking commands. These devices provide physical interaction with the system and allow the driver to generate continuous control inputs during teleoperation.

For immersive perception and camera control, a Meta Quest 3 VR headset is used. The headset provides head-tracking information that is mapped to camera pan–tilt motion on the robot, enabling natural viewpoint control during operation. The use of VR allows the driver to control both vehicle motion and viewing direction simultaneously within the experimental setup.

These interface devices form a critical part of the DiL loop, as they directly influence how the human operator perceives the environment and how control commands are generated.

2.2 Robotic Vehicle Platform

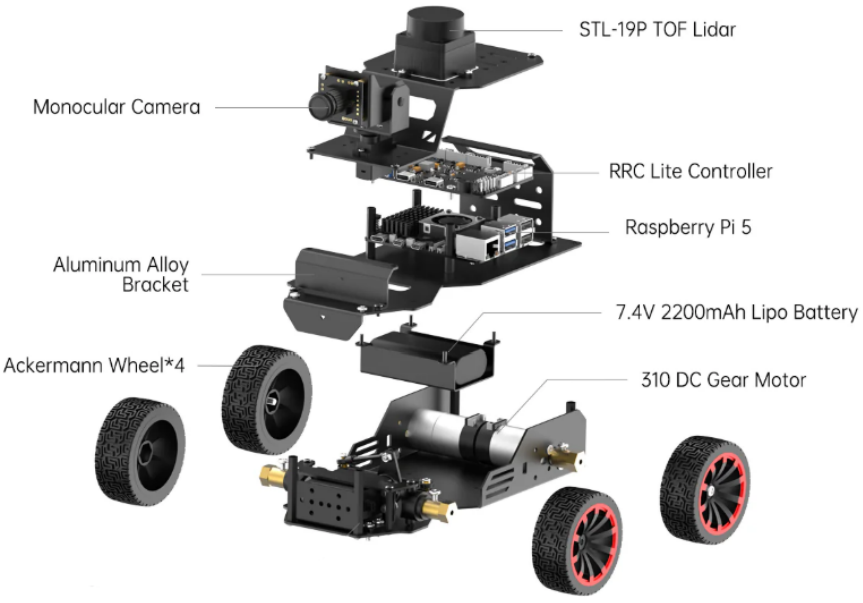


Figure 2.2: Components of the robot vehicle [7]

The prototyping platform in this project is built on the Mentor Pi robotic vehicle, see Figure 2.2. It is a small-scale, four-wheeled mobile robot equipped with a

Raspberry Pi single-board computer as its main controller. The vehicle provides an ideal foundation for testing perception, control and communication systems in a safe and flexible environment. The robot includes several key hardware components that make it suitable for DiL implementation, see Table 2.1 for component description and specifications. It executes control commands received through ROS 2 and provides live sensor data to the operator interface. This closed-loop configuration allows the system to simulate realistic vehicle dynamics and driver interaction while remaining safe and compact for laboratory use.

Table 2.1: Components of the robot platform and their specifications [7]

Component	**Function / Description**	**Specifications / Capabilities**
STL-19P TOF LiDAR	Provides 360° of distance measurements. Helps with obstacle and object detection	• Range up to $12m$ • Accuracy $\pm 2cm$ • Scan rate: $10Hz$
Monocular Camera	Camera with 2–DOF	• Resolution: 640x480 • Aperture: 2.0 • Focus: $1.7mm$ • Focal FOV angle: 170°
RRC Lite Controller	Handles motor control, PWM signals and sensor data	• Supports 4 motors and encoder feedback
Raspberry Pi 5	The main processing unit running ROS 2	–
LiPo Battery	The power source of the robot	• 7.4V • 2200mAh
Encoder Geared Motor ×2	Two DC geared motors with built-in encoders enabling independent drive of the rear wheels	–
Ackermann Wheels ×4	Enabling realistic steering kinematics	• Steering angle: ±30° • Wheel diameter:$65mm$

2.3 Robotic Operating System 2

ROS 2 is an open-source framework for building complex robotic systems, providing support for modular development, asynchronous communication, and scalable integration of sensing, control, and actuation [8].

2.3.1 Core Communication Abstractions

The core of ROS2 consists of Nodes and Topics, which together define the publish–subscribe communication structure of the system. Each node is responsible for a specific functional task, while topics serve as asynchronous data channels between nodes. The following table summarizes the core communication abstractions in ROS2, see Table 2.2.

Table 2.2: Overview of the ROS 2 communication architecture

Type	Description / Function
Node	A running program that performs a specific task in the system, such as sensing, control, or decision-making, and communicates with other nodes.
Publisher	A component inside a node that sends data messages on a topic, for example sensor measurements or control commands.
Subscriber	A component inside a node that receives data from a topic and reacts to new messages, typically by updating logic or triggering actions.
Topic	A named communication channel used to continuously stream data asynchronously between publishers and subscribers.
Message	A predefined data structure that specifies how information is formatted when exchanged between nodes.

2.4 Driver-in-the-Loop and HMI in ADAS Evaluation

As discussed in Chapter 1, many ADAS functions operate in a shared-control regime, where system support and human decision-making coexist. Such systems are designed to assist the driver continuously through information, warnings, or limited control actions, rather than replacing the driver entirely [6]. To evaluate these systems beyond purely technical performance, a Driver-in-the-Loop (DiL) approach is therefore required [4].

From a system perspective, DiL refers to a testing configuration in which a human operator is actively included in the vehicle control loop while interacting with ADAS functions in real time. Unlike fully automated or offline simulations, DiL

enables direct observation of how drivers perceive, interpret, and respond to safety-critical feedback, warnings, and partial interventions.

In the context of ADAS evaluation, DiL is particularly relevant for functions such as Automatic Emergency Braking (AEB), where driver awareness and reaction may influence both system effectiveness and acceptance. Warnings or preparatory cues must be correctly understood by the driver to avoid or mitigate a collision when full automation is not intended or available [6, 1].

2.4.1 Human-Machine Interface

The Human-Machine Interface (HMI) constitutes the primary communication channel between ADAS and the driver. A user interface comprises the visual and interactive elements through which a user interacts with a system [9]. Its role is to translate sensor data and system states into meaningful feedback that supports timely and appropriate driver action, especially for warning- and information-based assistance systems [6].

Within a DiL setup, the HMI is critical, as the driver relies entirely on mediated feedback to perceive the vehicle's environment and system intent. Effective HMI design can reduce cognitive load, shorten reaction times, and increase the likelihood that ADAS warnings result in correct human responses, such as braking or steering.

In this project, the HMI is designed to support ADAS evaluation rather than driver comfort or long-term usability. Key requirements therefore include low latency, clear warning presentation, and minimal visual clutter, ensuring that safety-relevant information is both noticeable and interpretable under time-critical conditions.

3
Methodology

The following section describes the methodology applied to integrate DiL into the robotic vehicle platform. It details the logic of each individual node, and further explains how the nodes interact, exchange information and work together to accomplish the intended objective.

The code repository is available on GitHub [10].

3.1 Node Structure

The node structure is organized around three main functional layers: vehicle control, information processing and HMI. Figure 3.1 presents an overview of the node structure and data flow within the platform. To improve readability, the architecture diagram uses color coding to distinguish between functional groups of nodes.

Nodes and connections shown in **red** represent the primary control loop responsible for vehicle motion. This includes driver input handling and low-level actuation such as steering, throttle, braking and gear control. These nodes form the core driving pipeline and directly influence the robot's movement.

Green nodes represent the information and perception logic of the system. This layer collects and processes sensor data and system state information, including LiDAR-based distance measurements and controller actuation data. This is where the ADAS functions are implemented. The processed information is aggregated and prepared for visualization before being forwarded to the HMI.

Blue nodes represent the HMI system. This subsystem is responsible for camera streaming, camera servo control and VR interaction. All VR-related components receive their system state information from the message node, ensuring that the VR interface reflects the same perception and control data as the standard camera stream.

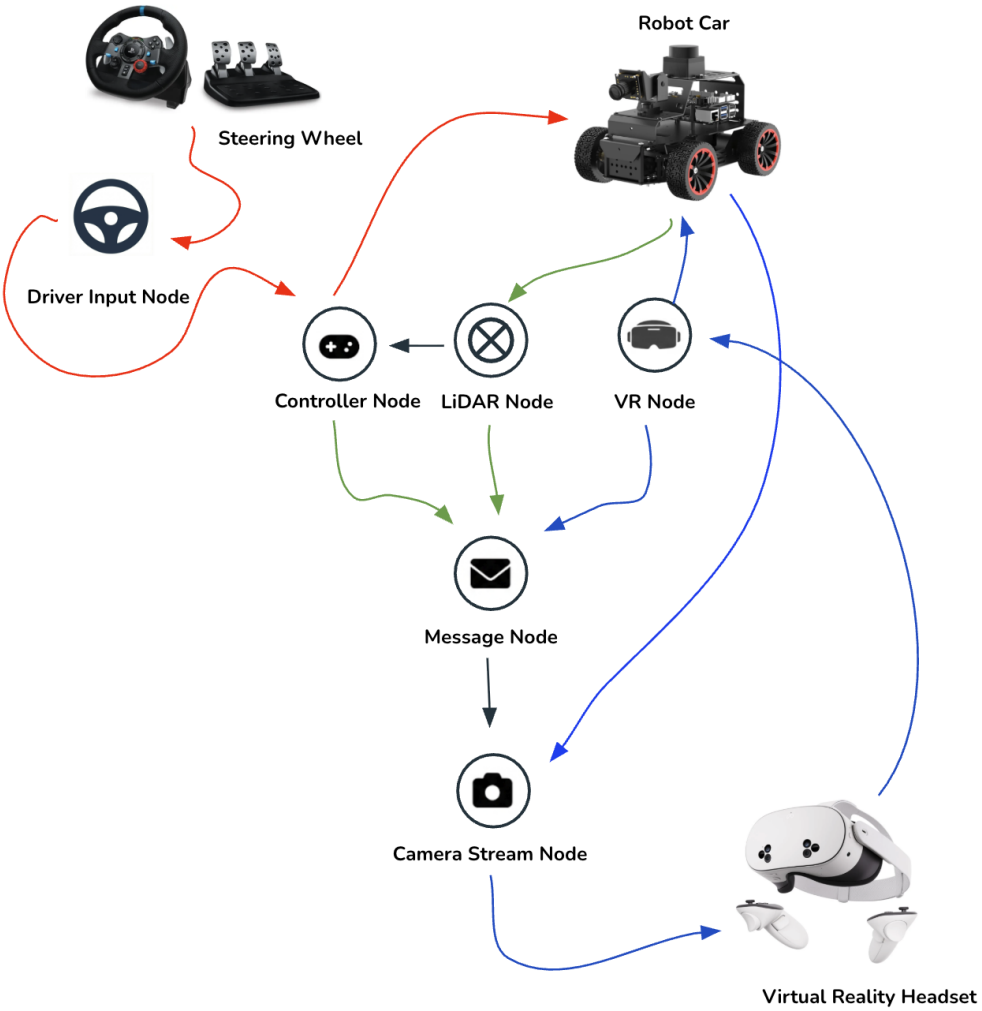


Figure 3.1: ROS 2 node architecture and information flow of the DiL platform.

3.2 Node Implementation

Within this architecture, functionality is distributed across multiple ROS 2 nodes, each responsible for a specific task. This modular design allows components to be modified or replaced without affecting the rest of the system, as the nodes work asynchronously. Table 3.1 summarizes the main nodes used in the implementation and their primary function in the node structure.

Table 3.1: Summary of implemented ROS 2 nodes and their primary responsibilities.

Node	Function
Driver Input Node	Reads steering wheel, throttle, brake and gear inputs and publishes normalized driver commands. [Running on separate PC]
Controller Node	Converts driver commands into motion control signals and publishes robot actuation commands.
LiDAR Node	Processes raw LiDAR scans and publishes minimum obstacle distance for the forward sector.
VR Node	Receives head tracking data from the VR headset and publishes camera pan-tilt commands.
Message Node	Aggregates perception and control state information and generates overlay data for visualization.
Camera Stream Node	Streams live video from the onboard camera and renders overlays for the operator interface.

3.2.1 Driver Input Node

The Driver Input Node is launched on a separate PC and acquires raw driver commands from the steering wheel and pedal interface. Inputs include steering angle, throttle position, brake position and discrete gear selection.

All inputs from the Logitech steering wheel are normalized to a common representation that the robot understands before being forwarded to the controller node. Steering is mapped to a dimensionless command

$$u_{\text{steer}} \in [-1,\, 1],$$

where negative values represent left steering and positive values represent right steering. Throttle and brake inputs are mapped to

$$u_{\text{throttle}},\; u_{\text{brake}} \in [0,\, 1].$$

To avoid unintended actuation, small input variations around zero are suppressed using a deadzone. Throttle and brake are combined such that braking always reduces propulsion:

$$u_{\text{drive}} = u_{\text{throttle}}(1 - u_{\text{brake}}).$$

Gear selection is handled through paddle inputs and determines whether forward motion, reverse motion, or neutral is allowed. In neutral gear, drive commands are suppressed.

The final command vector

$$(u_{\text{drive}},\, u_{\text{steer}},\, g)$$

is transmitted to the controller node at a fixed update rate. By separating driver input acquisition from vehicle control, the system allows alternative input devices to be integrated without modifying the actuation logic.

Communication between the Driver Input Node and the Controller Node is implemented using the User Datagram Protocol (UDP). UDP is chosen due to its low latency and minimal communication overhead, which are critical for real-time vehicle control. Because driver commands are transmitted at a high update rate and only the most recent command is relevant for actuation, occasional packet loss does not significantly affect system performance.
The command vector

$$(u_{\text{drive}},\, u_{\text{steer}},\, g)$$

is serialized and sent as a UDP packet at a fixed rate. The Controller Node continuously listens for incoming packets and always applies the most recently received command. This stateless communication scheme avoids blocking behavior and ensures that outdated commands are naturally discarded in favor of newer inputs.
This communication design preserves the modular structure of the system and allows the Driver Input Node to execute on a separate machine without introducing tight timing dependencies in the control loop.

3.2.2 Controller Node

The Controller Node converts normalized driver commands into motion commands for the robotic vehicle. It acts as the interface between the driver input layer and the low-level actuation system.

The longitudinal velocity command is computed from the drive input and a configurable maximum speed:

$$v = u_{\text{drive}}\, v_{\max}.$$

Steering is implemented using an Ackermann-like kinematic approximation. The steering command is converted into a yaw rate based on the vehicle wheelbase L:

$$\omega = \frac{v}{R}, \qquad R = \frac{L}{\tan(\delta)},$$

where δ is the steering angle.

The resulting linear and angular velocities are published as `geometry_msgs/Twist` messages on the control topic.

The controller also integrates safety constraints by using the ETTC metric to determine the activation of the AEB system [5, 1]. When AEB is active, forward propulsion is regulated and may be suppressed depending on the current speed and

stopping distance estimate. The logic computes a speed-dependent stopping distance based on the robot's current longitudinal velocity, an assumed reaction time (including sensing and processing delays), and the maximum achievable deceleration of the platform:

$$d_{\text{stop}} = v_{\text{robot}}\, t_{\text{reaction}} + \frac{v_{\text{robot}}^2}{2a_{\text{max}}}.$$

The final AEB trigger distance is obtained by augmenting the computed stopping distance with a fixed LiDAR sensor offset and a safety margin. In the implemented system, the reaction time is set to $t_{\text{reaction}} = \mathbf{0.25\,s}$ and the maximum deceleration is assumed as $a_{\text{max}} = \mathbf{1.0\,m/s^2}$. A sensor offset of $\Delta d_{\text{sens}} = \mathbf{0.15\,m}$ is added to compensate for the distance between the LiDAR sensor and the front bumper, while the safety margin is set to $\Delta d_{\text{safe}} = \mathbf{0.00\,m}$ in the current implementation. The resulting trigger distance is therefore given by

$$d_{\text{trigger}} = d_{\text{stop}} + \Delta d_{\text{sens}} + \Delta d_{\text{safe}}.$$

Additionally, a minimum trigger distance of **15 cm** is enforced to avoid unrealistically small thresholds at low speeds. Emergency braking is activated whenever the measured front obstacle distance d_{min} falls below this threshold.

When the AEB system is active, forward propulsion is not strictly disabled but regulated according to the current vehicle speed and the stopping distance estimated by the ETTC logic. At higher speeds, forward motion is suppressed earlier to ensure sufficient braking margin, while at lower speeds the driver is allowed to continue approaching the obstacle more slowly before the braking condition is met. Reverse motion remains fully permitted at all times, allowing the driver to retreat from hazardous situations and reposition the vehicle safely. Throughout AEB intervention, all perception, control and visualization functions remain operational, ensuring continuous situational awareness and system responsiveness.

3.2.3 LiDAR Node

The LiDAR Processing Node extracts proximity information from raw laser scan data. It subscribes to `sensor_msgs/LaserScan` and evaluates distance measurements within a forward-facing sector of ±45°, as illustrated in Figure 3.2.

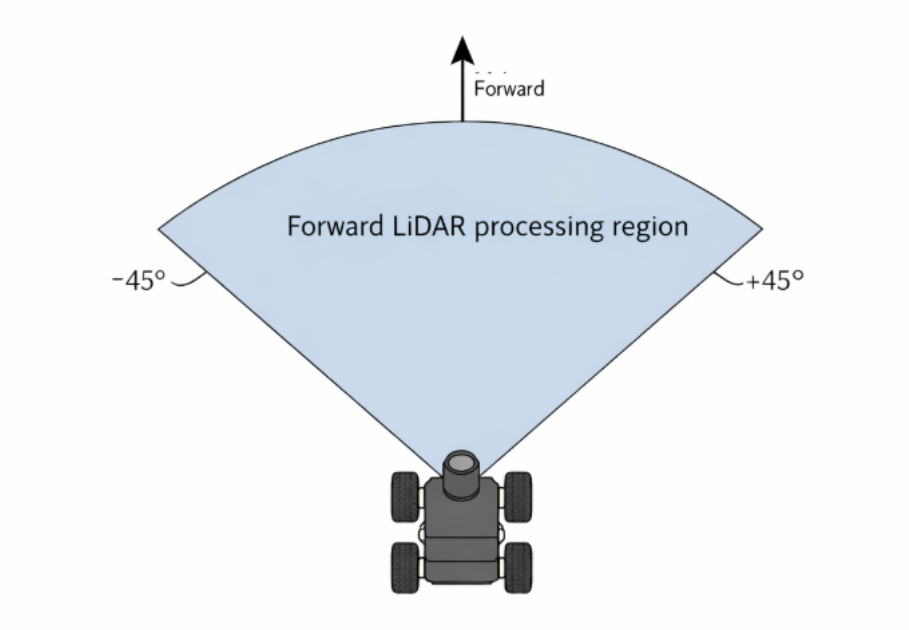


Figure 3.2: Forward-facing LiDAR processing sector (±45°) used for obstacle evaluation.

Only valid range measurements within the sensor limits are considered. From this set, the minimum distance in front of the robot is computed:

$$d_{\min} = \min(r_i),$$

and published as a scalar value in centimeters.

Restricting processing to the forward sector prevents obstacles behind the vehicle from triggering safety functions during reversing maneuvers and improves teleoperation robustness [3].

3.2.4 VR Node

The VR Node enables immersive camera control using head-tracking data from a VR headset. Orientation data are received over UDP and interpreted as camera pan and tilt commands.

The received angles are limited to a predefined range and mapped linearly to servo control signals:

$$\text{PWM} = C + \frac{\alpha}{\alpha_{\max}} R,$$

where C is the servo center position and R is the servo range.

The resulting commands are published to the robot controller to actuate the camera servos. For visualization and debugging, the applied servo values are also published as diagnostic data.

3.2.5 Message Node

The Message Node acts as the aggregation layer for system state information. It subscribes to perception and control topics, including LiDAR distance, gear state, camera orientation and emergency braking status.

The received data are converted into compact, human-readable overlay texts. Emergency braking is displayed as a warning only when active, while distance, gear and camera orientation are continuously updated.

This design separates visualization formatting from control and perception, ensuring that the operator interface always reflects the latest system state.

3.2.6 Camera Stream Node

The Camera Stream Node provides real-time visual feedback to the operator. It subscribes to the onboard camera image topic, overlays system information provided by the message node and streams the annotated video to external clients.

The processed frames are encoded as JPEG images and served as a live MJPEG stream, with an additional snapshot endpoint for single-frame access. Overlay information includes obstacle distance, gear state, camera orientation and emergency braking warnings.

By operating independently of the control pipeline, the camera stream remains active even when motion commands are suppressed, supporting safe and continuous DiL operation.

3.3 Human–Machine Interface

This section describes the implementation of the HMI within the DiL platform, with a focus on how the VR mode is integrated into the system. In particular, it explains how the VR client obtains the live camera stream from the robot and how head tracking is used to control the robot-mounted pan–tilt camera. The design goal is that both a standard browser client and the VR client observe the same video and system state information.

3.3.1 Overall Architecture and Network Requirements

The HMI is implemented as a distributed system consisting of a vehicle side (Raspberry Pi mounted on the robot) and an operator side (web browser or Meta Quest 3 VR headset). During operation, the VR headset connects to the robot's Wi-Fi hotspot, ensuring that the operator and robot are located within the same local network.
Two primary communication channels are used:

- **HTTP**: for transmitting the live camera image stream from the robot to the operator.
- **UDP**: for transmitting head-tracking data from the operator to the robot for real-time camera pan–tilt control.

This architecture was designed to support low-latency bidirectional communication while keeping the visualization and control subsystems decoupled from the core vehicle control logic.

3.3.2 HTTP Video Service

On the robot, the `stream.py` node subscribes to the ROS 2 camera topic, converts the messages to OpenCV frames, encodes them as JPEG images and starts an HTTP server on port `5000`. Two endpoints are provided:

- `/video_feed`: continuous MJPEG stream for web browser clients.
- `/snapshot`: single-frame JPEG endpoint optimized for VR clients.

In VR mode, the Unity application uses the `/snapshot` endpoint instead of maintaining a long-lived MJPEG connection. The client periodically requests `http://192.168.149.1:5000/s` decodes the received JPEG byte array into a texture, and applies it to a virtual display surface inside the Unity scene. This polling-based approach allows stable refresh behavior and controlled bandwidth usage on the embedded platform.

3.3.3 System State Overlays

To improve situational awareness, key system states are overlaid onto the live camera image before streaming. A dedicated message node subscribes to the latest perception and control signals, including front obstacle distance, gear state, emergency braking status, and camera pan–tilt feedback, and converts them into compact text labels.
The streaming node renders these overlays directly into each outgoing video frame. The overlay layout is fixed for readability: gear is shown in the top-left, pan/tilt in the top-right, and the measured distance at the bottom center. When AEB is active, a prominent warning is displayed in the image center, as illustrated in Figure 3.3. Since the overlay is generated on the robot side, both the browser view and the VR view receive identical annotated video without requiring client-side synchronization.

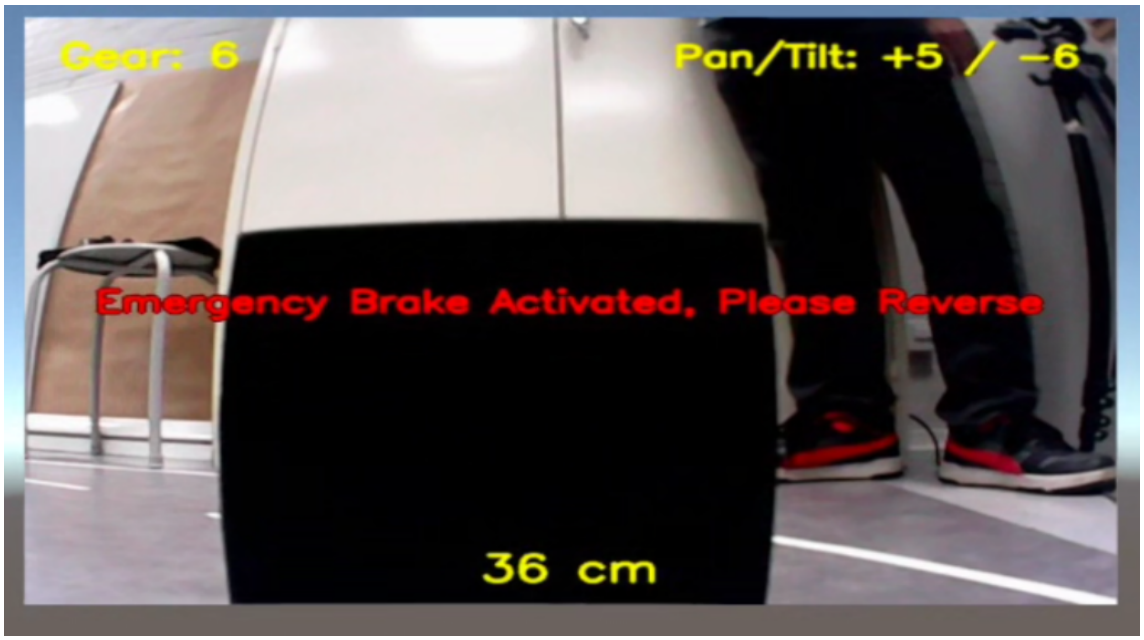


Figure 3.3: Annotated camera stream showing system state overlays: gear (top-left), camera pan/tilt (top-right), measured front distance (bottom center), and AEB warning displayed in the image center.

3.3.4 VR Camera Control

On the VR side, Unity continuously reads the headset orientation expressed as Euler angles and extracts the pitch and yaw components. These angles are normalized and transmitted as a compact UDP message in the format

`P:<pitch>,Y:<yaw>`

to `255.255.255.255:5005`.
On the robot side, a UDP receiver listens on port `5005`, parses the incoming values, applies safety bounds (typically clamped to $\pm 90°$), and maps the bounded angles linearly to PWM commands (1000–2000, centered at 1500) for the servo controller. The resulting signals actuate the physical pan–tilt camera mount. The applied PWM values are also published back into the ROS 2 system and reused by the visualization layer, ensuring that the displayed pan/tilt indicators remain consistent with the actual camera pose.
This closed-loop control scheme enables intuitive head-tracked camera operation while maintaining consistent system state visualization for both VR and browser clients.
During experiments, the video service was found to be most stable when accessed by a single client at a time. Concurrent requests from multiple devices may reduce refresh rate or affect connection stability. Therefore, the experimental setup used a single active viewer (browser or VR client) at any given time to ensure reliable operation.

3.4 Testing & Validation

The proposed DiL platform was validated through a set of scenario-based tests conducted in a controlled indoor laboratory environment. The purpose of the validation was not to evaluate optimal driving performance, but to verify correct system operation and interaction between perception, control, and visualization components under representative DiL use cases.
All tests were performed using the same robot platform, sensor configuration, and ROS 2-based control architecture described in previous sections. To ensure repeatability and safe operation, a fixed test track and static obstacles were used throughout the validation process.
Two operating configurations were evaluated:

- A baseline configuration using standard camera-based visual feedback.
- A VR configuration using a head-mounted display with head-tracked camera pan–tilt control.

Within these configurations, continuous driving, maneuvering, and emergency braking scenarios were executed to exercise the core platform functionalities, including camera-based teleoperation, LiDAR-based obstacle awareness, camera orientation control, and example ADAS behavior.

3.4.1 Teleoperation and Robot Maneuvering Test

A continuous teleoperation scenario was executed in which the robot was driven around a closed-loop oval track using only the streamed camera view.
The physical test environment and baseline teleoperation scenario are shown in Figures 3.4 and 3.5.

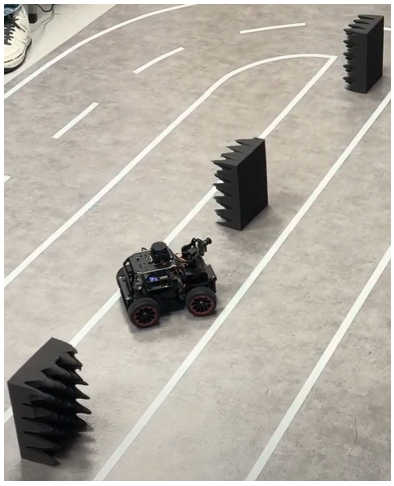


Figure 3.4: Indoor test track used for continuous teleoperation and obstacle maneuvering

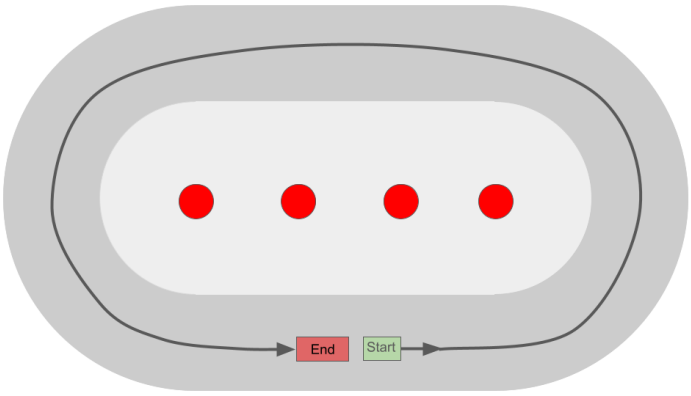


Figure 3.5: Schematic overview of the baseline teleoperation scenario

3.4.2 Obstacle Maneuvering and AEB scenario

A combined scenario consisting of slalom maneuvering and AEB was executed in the HMI configuration. A schematic overview of the combined scenario and the emergency braking setup are shown in Figures 3.6 and 3.7.

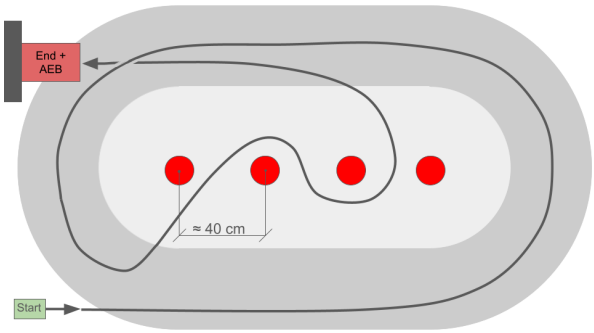


Figure 3.6: Schematic overview of the combined VR driving scenario.

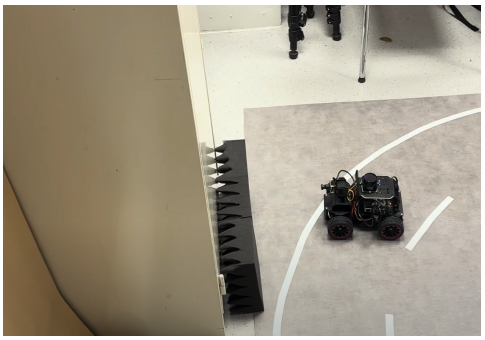


Figure 3.7: Physical setup for the AEB test with a protective barrier.

3.5 Test Repetition and System Reset

To evaluate system stability and repeatability, different reset strategies were applied depending on the test scenario.

During continuous teleoperation experiments without VR, the robot was operated for extended periods without restarting the ROS 2 system. This approach was used to assess long-term stability, communication robustness, and the ability of the platform to maintain consistent control and sensor streaming under sustained operation. Any observed degradations, such as delayed responses or control irregularities, were noted as indicators of accumulated system load or communication issues.

For the combined HMI and robot maneuvering scenario, the system was restarted between each test run. A full restart ensured consistent initial conditions, including camera orientation, servo offsets, network connections, and ROS 2 node states. This procedure minimized the influence of residual states from previous runs, such as accumulated servo drift or temporary communication delays, thereby improving the comparability of results across repeated HMI tests.

By applying scenario-specific reset strategies, the testing methodology balances realism and experimental control. Continuous operation reflects real teleoperation use cases, while controlled restarts support repeatable evaluation of HMI-based Driver-in-the-Loop performance.

4
Results

This chapter presents the outcomes obtained during testing and validation of the implemented DiL platform. The results focus on functional capabilities, operational stability and observed system limitations.

A documentation of results noted in a spreadsheet are presented in the Appendix, see Appendix A.1. In the spreadsheet the test, the desired outcome, number of successful runs, number of failed runs, accuracy rate and comments of the outcome is presented.

4.1 Continuous Teleoperation

The platform enabled real-time manual driving using steering, throttle, braking, and gear selection inputs. Vehicle motion commands were applied continuously in response to driver input, which were successfully completed.

In the baseline configuration without the HMI subsystem, the robot was driven continuously around the oval track using camera-based visual feedback only on the computer. The system completed twenty consecutive laps without interruption or system restart.

Steering, throttle and braking inputs remained responsive throughout the test, and the camera stream provided stable visual feedback. Table 4.1 presents the result of the continuous teleoperation test.

Table 4.1: Results from the continuous teleoperation test without VR.

Metric	**Result**
Total laps executed	20
Completed laps	20
Interrupted runs	0
System restarts	None

4.2 HMI performance

This section presents the results of the implemented HMI with focus on camera control and visual feedback during DiL operation.

4.2.1 Camera control and Interaction

The project outcome successfully enabled real-time camera control during both standard and VR-based operation. Camera pan and tilt were controlled either through conventional input devices (such as the joystick) or head-tracking data from the VR headset. The applied camera orientation was continuously reflected in the visual overlay, allowing the driver to maintain awareness of the current viewing direction. During testing, camera motion remained responsive and stable, with low noticeable lag between user input and visual feedback.

4.2.2 Visual Feedback and Safety Overlay

The visual output of the HMI during VR operation is shown in Figure 3.3. The annotated camera stream combines live video from the onboard camera with overlaid system and safety information, providing the driver with real-time situational awareness.

The overlay includes the current gear state, camera pan and tilt angles, and the LiDAR-based distance to the nearest forward obstacle. When the AEB function is triggered, a prominent warning message is displayed at the center of the visual field. In the example shown, AEB is activated at a measured obstacle distance of 36 cm, and the driver is instructed to reverse.

By integrating control state, perception data, and safety warnings directly into the camera stream, the HMI ensures that critical information is immediately visible without requiring the driver to shift attention away from the driving task. This unified visualization was found to be particularly effective during VR operation, where the video stream serves as the sole visual input to the driver.

4.3 Combined HMI and Robot Maneuvering Scenario

In the VR configuration, a combined scenario consisting of continuous driving, slalom maneuvering and AEB was executed.

Across five runs, four were completed without interruption. In completed runs, the robot completed one lap, navigated the slalom section and triggered emergency braking during the wall-approach stage.

Table 4.2: Results from the combined VR driving scenario.

Metric	Result
Total runs	5
Completed runs	4
Failed runs	1
Observed failure mode	Paralyzed system state
Success rate	80%

With the use of ETTC, the AEB activation was no longer based on a fixed distance threshold. Instead, emergency braking was triggered according to the current vehicle velocity and the estimated stopping distance. At higher speeds, the braking intervention occurred at a larger distance to the obstacle, while at lower speeds the robot was allowed to approach closer before AEB activation.

This behavior resulted in more consistent and realistic braking responses across different driving conditions. During the combined scenario, the AEB was observed to activate reliably during the wall-approach stage, preventing collision while avoiding unnecessary early braking during low-speed maneuvering.

4.4 Observed "Paralyzed" System State

During one test run with the HMI enabled, a paralyzed system state was observed. During this state, sensor data acquisition and visualization remained active, including the live camera stream, LiDAR distance updates, gear indicators and emergency brake warnings. However, vehicle motion and camera orientation no longer responded to driver input.

If the paralyzed state occurred while the robot was moving, it continued at a constant speed. The state persisted until the control software or the robot system was restarted, after which normal operation was restored. This behavior was not observed in any of the test runs without VR.

5
Discussion

This chapter discusses the obtained results in relation to the intended purpose of the project and the limitations observed during testing. Rather than drawing definitive conclusions, the discussion focuses on the validity of the performed validation activities, the scope of the implemented functionality and the implications of the observed system behavior.

5.1 Test Scope and Limitations

The conducted validation activities demonstrate that the robot platform is capable of executing the defined driving scenarios under controlled indoor conditions. However, the limited number of repetitions and the constrained variety of scenarios restrict the strength of any claims regarding overall system stability or robustness.

While repeated successful runs indicate that the platform can operate reliably within the tested conditions, the results should be interpreted as indicative rather than conclusive. The performed tests were not designed to statistically quantify stability margins, failure rates, or long-term reliability under varying environmental or operational conditions. As a result, the validation primarily demonstrates functional feasibility rather than providing comprehensive evidence of robustness for all intended DiL applications. This distinction between functional feasibility and comprehensive robustness evaluation is consistent with the role of early-stage DiL platforms described in the literature [4].

This limitation becomes particularly relevant when considering more complex interactions involving higher system load, tighter safety margins, or extended operation time, where untested edge cases may become dominant.

5.2 Validation and System Applicability

The implemented platform focuses on a single, simplified ADAS in the form of AEB. While this function demonstrated perception-based safety intervention within the tested scenarios, it represents only a narrow subset of typical ADAS functionality. Other common assistance systems, such as Lane Departure Warning, Rear Collision Warning, or adaptive longitudinal control, were not implemented or evaluated within the scope of this project. This reflects common early-stage ADAS prototyping approaches, where individual safety functions are isolated and evaluated before

more complex multi-function systems are considered [1, 3].

In addition, although a structured validation methodology was defined, the extent of practical testing was limited by the available project timeframe. As a result, only a subset of the planned scenarios and repetitions could be executed. Consequently, the results should be interpreted primarily as a validation of the platform architecture and overall system feasibility rather than as a comprehensive assessment of ADAS performance.

The modular ROS 2-based design suggests that additional ADAS functions could be integrated into the platform. However, the behavior of such functions, their interaction with human drivers and potential combined system effects remain unexplored and would require more extensive validation.

From an application perspective, the platform performed reliably for simpler tasks such as continuous driving without VR, where more than twenty consecutive laps were completed consistently. This indicates that the system is well suited for educational use and for demonstrating fundamental ADAS and DiL concepts, particularly in Hardware-in-the-Loop or DiL teaching environments. The observed speed-dependent braking behavior aligns with collision risk modeling approaches that incorporate stopping distance and reaction time rather than fixed distance thresholds [5].

Whether the added complexity of VR provides sufficient value on a platform of this scale remains an open question. While VR offers increased immersion, its benefits must be carefully weighed against the increased system complexity and the reduced robustness observed during testing.

5.3 System Limitations

Several limitations affecting the platform were identified during development and validation, primarily related to system integration and operational complexity. The most significant limitation was the occurrence of a paralyzed system state during operation with VR enabled.

During this state, sensor data acquisition and visualization continued to function as expected, including the live camera stream, LiDAR distance updates, gear indicators and AEB warnings. However, driver commands were no longer applied to vehicle motion or camera orientation, preventing effective control of the robot.

Repeated debugging efforts and discussions with more experienced developers did not reveal any direct faults in the project-specific control code developed within this work. The observed behavior instead appears to be related to interactions with pre-existing system components or background services provided by the robot platform. Such interactions may interfere with control authority when multiple subsystems operate concurrently.

In addition to control-related limitations, the system relies on a software-processed video stream, where image post-processing and re-encoding introduce a consistent transmission latency. This latency prevents instantaneous system response and introduces a delay in the perception–action loop, limiting the suitability of the platform for highly dynamic DiL scenarios. Within the VR configuration, the delayed visual feedback increases sensory mismatch between visual input and physical motion, which may contribute to motion sickness or cybersickness during prolonged operation. Such effects are well documented in the context of HMI, where increased latency and sensory mismatch are known to negatively impact operator performance and user experience [6, 9].

As these issues could not be resolved within the project timeframe, it limits the current stability and reliability of the VR configuration. Further investigation and architectural refinement are required before VR-based control can be considered suitable for consistent and robust DiL operation.

6
Future Developments

This section outlines potential directions for future work that could extend the capabilities of the proposed DiL platform. The listed developments represent feasible research directions rather than finalized implementation plans and would require further investigation and validation.

6.1 Sensor and Perception

Further development within sensor and perception systems could significantly improve the fidelity and reliability of ADAS evaluation.

- **Sensor Deployment and Sensor Fusion**: One possible direction for future work is the investigation of expanded sensor configurations with higher resolution and complementary sensing modalities, including additional LiDAR and vision-based sensors. In combination, this could motivate the study of multi-sensor fusion algorithms capable of processing heterogeneous sensor data to improve perception robustness and ADAS decision-making accuracy.

- **SLAM-Based Localization and Mapping**: With enhanced sensing capabilities, the integration of SLAM (Simultaneous Localization and Mapping) methods could be explored. Such approaches may enable more accurate localization and environment representation within the controlled test environment, supporting repeatable and spatially consistent DiL experiments.

6.2 Improved Driver Immersion and HMI Integration

Improvements in driver immersion and HMI represent another potential extension of the platform, particularly for studies focusing on driver behavior within a DiL context.

- **Enhanced Virtual Environment Realism**: Future work could investigate increased realism in the VR environment to strengthen the DiL experience. This may include utilizing perception or SLAM-based map data to create more accurate and consistent virtual representations of the physical test environment.

- **Advanced HMI Concepts**: The feasibility of more advanced HMI solutions could be evaluated, such as digital cockpit interfaces or augmented reality head-up displays. These interfaces may improve situational awareness by presenting vehicle state and ADAS information in a more intuitive and context-aware manner.

6.3 Platform Scalability and Performance Optimization

Several limitations identified during development suggest possible future work related to scalability, performance, and system robustness of the DiL platform.

- **Vehicle Platform and Mechanical Scaling**: Future investigations could consider alternative robotic platforms with improved mechanical properties or increased scale. Such platforms may provide enhanced dynamic similarity to full-scale vehicles and support more representative ADAS and DiL experiments.

- **Computational Resources**: The impact of increased onboard computational power could be studied to assess its influence on system responsiveness and control complexity within the ROS-based architecture. This may enable experimental evaluation of more computationally demanding control strategies for ADAS validation.

- **Communication and System Stability**: Further work could focus on analyzing and mitigating communication latency and stability issues within the ROS-based system. In particular, systematic debugging and architectural refinement may help identify the root causes of observed system freezes or paralysis, improving overall robustness of the DiL platform.

7
Conclusions

This project has demonstrated the feasibility of constructing a small-scale, DiL ADAS prototyping platform using a robotic vehicle and a ROS 2-based modular software architecture. The implemented system successfully integrates human driver input, onboard perception, vehicle control and visual feedback into a closed-loop experimental platform.

The platform enabled stable remote teleoperation using camera-based visual feedback and physical driver controls. A simplified AEB function based on LiDAR distance measurements was implemented and verified under controlled indoor conditions. The modular node-based structure allowed perception, control and visualization components to operate independently, supporting flexible system modification and extension.

Testing results indicate that the system performs reliably during baseline teleoperation scenarios without VR, as demonstrated in the initial experiments using a fixed camera stream without camera movement control. When VR-based camera control was introduced, the increased system complexity led to reduced system stability, including the occurrence of a paralyzed system state during operation. These observations limit the current applicability of the VR configuration and highlight the importance of robust integration when combining multiple real-time subsystems. Based on the observed behavior, ADAS testing is more effectively conducted without VR in the system's present state, as the stability degradation introduced by VR outweighs the additional perceptual benefits it provides.

This outcome directly addresses the project objective defined in Chapter 1, which was to design and validate a modular DiL ADAS prototyping platform for controlled experimental use.

Overall, the project fulfills its primary objective of establishing a functional DiL ADAS prototyping platform. Although not intended as a production-ready system, the platform provides a valuable foundation for educational use and early-stage evaluation of HMI and basic ADAS concepts.

Bibliography

[1] L. Yang, Y. Yang, G. Wu, X. Zhao, S. Fang, X. Liao, R. Wang, and M. Zhang, "A systematic review of autonomous emergency braking," *Journal of Advanced Transportation*, 2022, accessed: 2026-01-11. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1155/2022/1188089

[2] Wikipedia contributors. (2026) Advanced driver-assistance system. Accessed: 2026-01-11. [Online]. Available: https://en.wikipedia.org/wiki/Advanced_driver-assistance_system

[3] A. M. A. Dr.Bassim Abdulbaqi Jumaa, Anwaar Mousa Abdulhassan. (2019) Advanced driver assistance system (adas): A review of systems and technologies. Accessed: 2026-01-11. [Online]. Available: https://www.scribd.com/document/667359079/Advanced-Driver-Assistant-System-ADAS-A-Review-of-System-and-Technologies

[4] P. Morse. (2018) Adas simulation technology: Keeping drivers 'in the loop'. Accessed: 2026-01-11. [Online]. Available: https://www.epdtonthenet.net/article/162407/ADAS-simulation-technology--keeping-drivers--in-the-loop-

[5] P. Goudarzi and B. Hassanzadeh, "Collision risk in autonomous vehicles: Classification, challenges, and open research areas," *Safety*, vol. 6, no. 1, 2024, accessed: 2026-01-11. [Online]. Available: https://www.mdpi.com/2624-8921/6/1/7

[6] M. Lu, K. Wevers, and R. Van der Heijden, "Technical feasibility of advanced driver assistance systems (adas) for road traffic safety," *Transportation Planning and Technology*, vol. 28, no. 3, pp. 167–187, 2005, accessed: 2026-01-11.

[7] HiWonder Robotics. (2025) Hiwonder mentorpi a1 product page. Accessed: 2025-11-03. [Online]. Available: https://www.hiwonder.com/products/mentorpi-a1?variant=41285898600535

[8] Open Robotics. (2022) Ros 2 humble hawksbill documentation. Accessed: 2026-01-11. [Online]. Available: https://docs.ros.org/en/humble/index.html

[9] Wikipedia contributors. (2026) User interface. Accessed: 2026-01-11. [Online]. Available: https://en.wikipedia.org/wiki/User_interface

[10] A. Chouhan, L. Fäldt, E. Jonsson, F. Liao, J. Wang, and P. R. Sridharraju. (2025) Code repository for driver-in-the-loop adas prototyping platform. Accessed: 2025-11-03. [Online]. Available: https://github.com/elliotjonsson/Codefiles-for-AEP-HIL-project/

A

Appendix

The following appendix contains supplementary material referenced throughout the report.

A.1 Testing log

Scenario based testing log									
Test number	**Test discription**	**Nodes required**	**Required nodes active?**	**Program desired outcome**	**Number of succeful run**	**Number of failed run**	**Success/ accuracy rate**	**Comments**	**Status**
1	Drive robot without physical vision, only camera stream for 20 laps around vinyl track	wheel_remote_control_brake.py	yes	receive the motion commands from G27.	5	0	1	robot looses communication when the input command rate is high, restarting package solves this most of the times. Ran on **macbook** which had stable connection	good
		g29_client_mac.py	yes	communication between laptop and G27 for robot					
		stream.py	yes	access the camera feed and stream to					
		message_node.py	yes	send messages with distances etc to other nodes					
2.1	Drive robot without physical vision, use VR environement and navigate a slalom track, then drive straight into an obstacle and have the system interfere and emergency brake	wheel_remote_control_brake.py	yes	receive the motion commands from G27.	1	4	0.2	Worked 1/5 times. had some input lag to the stream but was controllable once used to it.	needs improvement
		g29_client_mac.py	yes	communication between laptop and G27 for robot					
		stream.py	yes	access the camera feed and stream to					
		message_node.py	yes	send messages with distances etc to other nodes					
		VR_listener.py	yes	Control camera servo based of VR IMU					
2.2	Drive robot without physical vision, use VR environement and navigate a slalom track, then drive straight into an obstacle and have the system interfere and emergency brake	gen_steering.py	yes	receive the motion commands from G27.	4	1	0.8	Worked 4/5 times. had some input lag to the stream but was controllable once used to it. Much better connection	good
		gen_client.py	yes	communication between laptop and G27 for robot					
		stream.py	yes	access the camera feed and stream to					
		message_node.py	yes	send messages with distances etc to other nodes					
		VR_listener.py	yes	Control camera servo based of VR IMU					

TME180 Automotive Engineering Project 2025

Supervisor: Marco Dozza, Department of Mechanical Engineering
Supervisor: Rahul Rajendra Pai, Department of Mechanical Engineering,
Examiner: Alexey Vdovin, Department of Mechanical Engineering

Studentarbeten – Mekanik och maritima vetenskaper (M2) – Projektarbete
Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31 772 1000